МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет»
(национальный исследовательский университет)
Институт естественных и точных наук
Кафедра «Прикладная математика и программирование»

Рекомендательные системы

ОТЧЕТ
по лабораторной работе № 4
по дисциплине «Приложения и практика анализа данных»

Выполнил:
студент группы КЭ–305
_____ / А.С. Назарова /
«____» _____ 2024 г.

Проверил: ст. преподаватель
_____ / М.А. Ческидова /
«____»_____ 2024 г.

Челябинск – 2024

ОГЛАВЛЕНИЕ

1 ПОСТАНОВКА ЗАДАЧИ 2
2 ХОД РАБОТЫ 3
2.1 Гипотеза 1 9
2.2 Гипотеза 2 10
2.3 Гипотеза 3 12
2.4 Гипотеза 4 13
2.5 Гипотеза 5 15
2.6 Гипотеза 6 16
2.7 Гипотеза 7 17
2.8 Гипотеза 8 19
2.9 Гипотеза 9 21
2.10 Гипотеза 10 22
ВЫВОД 25
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 26

1 ПОСТАНОВКА ЗАДАЧИ

Требуется построить правило построения неизвестных рейтинговых оценок пользователя.

Цель данной лабораторной работы — построить и оценить рекомендательные системы для предсказания оценок пользователей шуткам. В качестве методов используются:

- Коллаборативная фильтрация (item-based CF);
- Матричная факторизация (SVD).

Тема «Шутки».

Набор содержит шутки и их оценки пользователями. Более 100 000 оценок от 7 699 пользователей: данные собраны с апреля 2015 г. по ноябрь 2019 г. Данные отформатированы как файл Excel, представляющий собой матрицу 7699 на 159, где строки — пользователи, а столбцы — шутки. Крайний левый столбец представляет количество шуток, оцененных каждым пользователем. Всего в этом наборе данных 7699 пользователей и 158 шуток. Каждая оценка находится в диапазоне от (- 10,00 до +10,00), а 99 соответствует нулевой оценке (пользователь не оценил эту шутку).

2 ХОД РАБОТЫ

Для начала нужно загрузить и обработать данные, разделить их на обучающую и тестовую выборки. Данные загружаются из Excel-файла с помощью pd.read_excel. Значение 99 обозначает отсутствие оценки. Мы заменяем его на 0, чтобы указать, что пользователь не оценил эту шутку.

Первый столбец (rated_count) содержит количество шуток, оцененных каждым пользователем. Остальные столбцы (ratings) представляют матрицу оценок.

```
import pandas as pd
import numpy as np

# Загрузка данных
df = pd.read_excel('April 2015 to Nov 30 2019.xlsx', header=None)

# Замена 99 на NaN
df.replace(99, 0, inplace=True)

# Отделение количества оцененных шуток и матрицы оценок
rated_count = df.iloc[:, 0]
ratings = df.iloc[:, 1:]
```

Создание копий данных:

- ratings_train — тренировочная выборка.
- ratings_test — тестовая выборка.

Для каждого пользователя случайным образом выбирается 20% его оценок и переносится в тестовую выборку. Оставшиеся оценки остаются в тренировочной выборке. В тренировочной выборке оценки, перенесенные в тестовую, заменяются на NaN, чтобы модель не использовала их при обучении.

```
ratings_train = ratings.copy()
ratings_test = pd.DataFrame(index=ratings.index, columns=ratings.columns)

for user in ratings.index:
    # Получаем индексы шуток, которые были оценены пользователем
    rated_jokes = ratings.columns[ratings.loc[user].notna()]

    # Проверяем, что пользователь оценил хотя бы одну шутку
    if len(rated_jokes) > 0:
        # Преобразуем rated_jokes в Series, чтобы можно было использовать
.sample()
        rated_jokes = pd.Series(rated_jokes)

        # Случайным образом выбираем 20% из оцененных шуток
```

```
        test_jokes = rated_jokes.sample(frac=0.2, random_state=42)

        # Переносим выбранные шутки в тестовую выборку
        ratings_test.loc[user, test_jokes.values] = ratings.loc[user,
test_jokes.values]

        # Убираем эти шутки из тренировочной выборки
        ratings_train.loc[user, test_jokes.values] = pd.NA
```

Вычисление сходства между шутками (item-based CF).

Вычисляется матрица сходства между шутками с использованием корреляции Пирсона. min_periods = 1 означает, что корреляция вычисляется даже для шуток, которые были оценены только одним пользователем.

```
# Вычисление сходства между шутками
item_similarity = ratings_train.T.corr(method='pearson', min_periods=1)
```

Функция предсказания для item-based CF.

Для заданной шутки (item_id) находятся шутки с наибольшим сходством. Если пользователь не оценил ни одной похожей шутки, возвращается средняя оценка пользователя. В противном случае вычисляется взвешенная сумма оценок похожих шуток.

```
# Функция предсказания
def predict_item_cf(user_id, item_id, ratings, similarity):
    # Получаем пользователей, которые оценили текущую шутку
    users_rated_item = ratings.loc[:, item_id].dropna()
    # Получаем сходство между текущей шуткой и другими шутками
    sim_scores = similarity.loc[item_id].dropna().drop(item_id,
errors='ignore')
    # Фильтруем похожие шутки, чтобы оставить только те, которые есть в
матрице ratings
    similar_items = sim_scores.index.intersection(ratings.columns)
    # Получаем оценки пользователя для похожих шуток
    user_ratings = ratings.loc[user_id, similar_items]
    # Если пользователь не оценил ни одной похожей шутки, возвращаем среднюю
оценку пользователя
    if user_ratings.isna().all():
        return ratings.loc[user_id].mean()
    else:
        # Вычисляем взвешенную сумму оценок
        weighted_sum = (user_ratings * sim_scores.loc[similar_items]).sum()
        sum_sim = sim_scores.loc[similar_items].sum()
```

```
        return weighted_sum / sum_sim if sum_sim != 0 else
ratings.loc[user_id].mean()
```

Оценка модели item-based CF.

RMSE — это метрика, которая измеряет среднее отклонение предсказанных значений от фактических. Чем меньше значение RMSE, тем лучше модель предсказывает оценки.

Вычисляется среднеквадратичная ошибка (RMSE) между предсказанными и фактическими оценками. Для каждой оценки в тестовой выборке предсказывается оценка с использованием item-based CF. RMSE используется для оценки точности модели.

```
# Функция для вычисления RMSE
def rmse(predictions, actual):
    return np.sqrt(((predictions - actual) ** 2).mean())

# Оценка модели
predictions = []
actual = []
for user in ratings_test.index:
    for item in ratings_test.columns:
        if pd.notna(ratings_test.loc[user, item]):
            pred = predict_item_cf(user, item, ratings_train,
item_similarity)
            predictions.append(pred)
            actual.append(ratings_test.loc[user, item])

rmse_value = rmse(np.array(predictions), np.array(actual))
print(f"RMSE для item-based CF: {rmse_value:.4f}")
```

Оценка модели item-based CF показала результат **RMSE = 1.6120**. Это хороший результат, так как он демонстрирует, что предсказанные оценки в среднем отличаются от фактических на 1.6120 балла Учитывая, что диапазон оценок составляет от -10 до +10, такое отклонение можно считать небольшим. Это подтверждает, что модель успешно учитывает сходство между шутками и способна делать достаточно точные предсказания.

Матричная факторизация (SVD).

Для работы SVD данные не должны содержать пропущенных значений. Мы заменяем NaN на 0. SVD разлагает матрицу оценок на латентные факторы пользователей и шуток.

SVD (Singular Value Decomposition - разложение по сингулярным значениям) — это метод, который позволяет представить матрицу оценок в виде комбинации более простых "строительных блоков". В контексте рекомендательных систем эти "строительные блоки" называются латентными факторами.

Латентные факторы пользователей — это наборы чисел, которые описывают предпочтения пользователей. Например, один фактор может показывать, насколько пользователь любит юмор, а другой — насколько он предпочитает сложные или простые шутки.

Латентные факторы шуток — это наборы чисел, которые описывают характеристики шуток. Например, один фактор может указывать на то, насколько шутка сложная, а другой — насколько она юмористична.

SVD разбивает матрицу оценок на эти латентные факторы, что позволяет:

- Упростить анализ данных.
- Предсказывать оценки, умножая факторы пользователей на факторы шуток.

Таким образом, SVD помогает понять, какие скрытые свойства (латентные факторы) влияют на оценки пользователей и шуток, и использовать это для создания рекомендаций.

```
from sklearn.decomposition import TruncatedSVD

# Замена NaN на 0 в ratings_train
ratings_train_filled = ratings_train.fillna(0)

# Применение SVD
svd = TruncatedSVD(n_components=10)
svd.fit(ratings_train_filled)  # Используем заполненную матрицу
user_factors = svd.transform(ratings_train_filled)
item_factors = svd.components_.T

# Функция предсказания для SVD
def predict_svd(user_id, item_id, user_factors, item_factors):
    user_vec = user_factors[user_id]
```

```
    item_vec = item_factors[item_id]
    return np.dot(user_vec, item_vec)

# Оценка модели для SVD
predictions_svd = []
actual = []  # Используем ту же переменную actual, что и для item-based CF
for user in ratings_test.index:
    for item in ratings_test.columns:
        if pd.notna(ratings_test.loc[user, item]):
            pred = predict_svd(user, item, user_factors, item_factors)
            predictions_svd.append(pred)
            actual.append(ratings_test.loc[user, item])

rmse_svd = rmse(np.array(predictions_svd), np.array(actual))
print(f"RMSE для SVD: {rmse_svd:.4f}")
```

Оценка модели SVD показала результат **RMSE = 1.7158**. Этот результат можно считать удовлетворительным, так как он показывает, что предсказанные оценки в среднем отличаются от фактических на **1.7158 балла**.

Сравнение результатов двух моделей:

- Item-based CF: RMSE = 1.6120;
- SVD: RMSE = 1.7158.

Item-based CF показала более низкое значение RMSE, что указывает на более высокую точность предсказаний по сравнению с SVD. Результат оценки SVD немного хуже, чем у item-based CF. Это может быть связано с тем, что SVD требует больше данных для более точного разложения матрицы, а также может быть чувствительна к пропущенным значениям.

Вывод рекомендаций для пользователей.

Функция get_recommendations для заданного пользователя предсказываются оценки для шуток, которые он еще не оценил. Рекомендации сортируются по убыванию предсказанных оценок. Для пользователя 0 выводятся рекомендации с использованием моделей item-based CF и SVD.

```
# Вывод рекомендаций для пользователей
def get_recommendations(user_id, ratings, model, top_n=5):
    """
    Получает рекомендации для пользователя на основе модели.
    """
    # Получаем шутки, которые пользователь еще не оценил
    unrated_jokes = ratings.columns[ratings.loc[user_id] == 0]

    # Предсказываем оценки для непросмотренных шуток
    predictions = {}
```

```
    for item in unrated_jokes:
        # Проверяем, что индекс шутки находится в допустимом диапазоне
        if item < len(ratings.columns):  # Убедитесь, что индекс не превышает
размер матрицы
            if model == 'item_cf':
                pred = predict_item_cf(user_id, item, ratings,
item_similarity)
            elif model == 'svd':
                pred = predict_svd(user_id, item, user_factors, item_factors)
            predictions[item] = pred
        # Если индекс выходит за пределы, просто пропускаем его

    # Сортируем по убыванию предсказанных оценок
    sorted_recommendations = sorted(predictions.items(), key=lambda x: x[1],
reverse=True)[:top_n]

    return sorted_recommendations

# Пример вывода рекомендаций для пользователя
user_id = 0  # Выберите ID пользователя
print(f"Рекомендации для пользователя {user_id} (item-based CF):")
recommendations_cf = get_recommendations(user_id, ratings_train, 'item_cf')
for joke, score in recommendations_cf:
    print(f"Шутка {joke}: Предсказанная оценка {score:.2f}")

print(f"\nРекомендации для пользователя {user_id} (SVD):")
recommendations_svd = get_recommendations(user_id, ratings_train, 'svd')
for joke, score in recommendations_svd:
    print(f"Шутка {joke}: Предсказанная оценка {score:.2f}")
```

```
Рекомендации для пользователя 0 (item-based CF):
Шутка 132: Предсказанная оценка 1.00
Шутка 26: Предсказанная оценка 0.88
Шутка 154: Предсказанная оценка 0.27
Шутка 63: Предсказанная оценка 0.25
Шутка 130: Предсказанная оценка 0.22

Рекомендации для пользователя 0 (SVD):
Шутка 71: Предсказанная оценка 3.68
Шутка 107: Предсказанная оценка 0.12
Шутка 118: Предсказанная оценка 0.06
Шутка 88: Предсказанная оценка 0.06
Шутка 55: Предсказанная оценка 0.04
```

Рисунок 1 – Вывод рекомендаций для пользователя 0.

Набор шуток отличается, потому что модели используют разные подходы для предсказания.

Item-based CF рекомендует шутки, которые похожи на те, что пользователь уже оценил. Модель использует сходство между шутками для предсказания

оценок. SVD рекомендует шутки на основе латентных факторов, которые описывают предпочтения пользователя и характеристики шуток.

Item-based CF рекомендует шутки с более высокими оценками (1.00, 0.88 и т.д.), что указывает на более высокую уверенность модели в релевантности этих шуток для пользователя 0.

SVD рекомендует шутку 71 с высокой оценкой (3.68), но остальные шутки имеют очень низкие оценки (0.12, 0.06, 0.04), что может свидетельствовать о том, что модель SVD менее уверена в их релевантности.

2.1 Гипотеза 1

Гипотеза 1: Пользователи, которые оценили больше шуток, получают более точные рекомендации.

Эта гипотеза предполагает, что чем больше шуток пользователь оценил, тем лучше система рекомендаций может предсказать его предпочтения и предоставить более точные рекомендации. Основная идея заключается в том, что большее количество оценок позволяет системе лучше понять вкусы пользователя, что, в свою очередь, должно приводить к более релевантным рекомендациям.

```
def hypothesis_1():
    # Количество оценок на пользователя
    user_ratings_count = ratings.apply(lambda x: x.count(), axis=1)

    # График
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=user_ratings_count, y=rated_count)
    plt.title('Гипотеза 1: Зависимость количества оценок от количества
рекомендаций')
    plt.xlabel('Количество оценок пользователя')
    plt.ylabel('Количество рекомендаций')
    plt.show()
hypothesis_1()
```

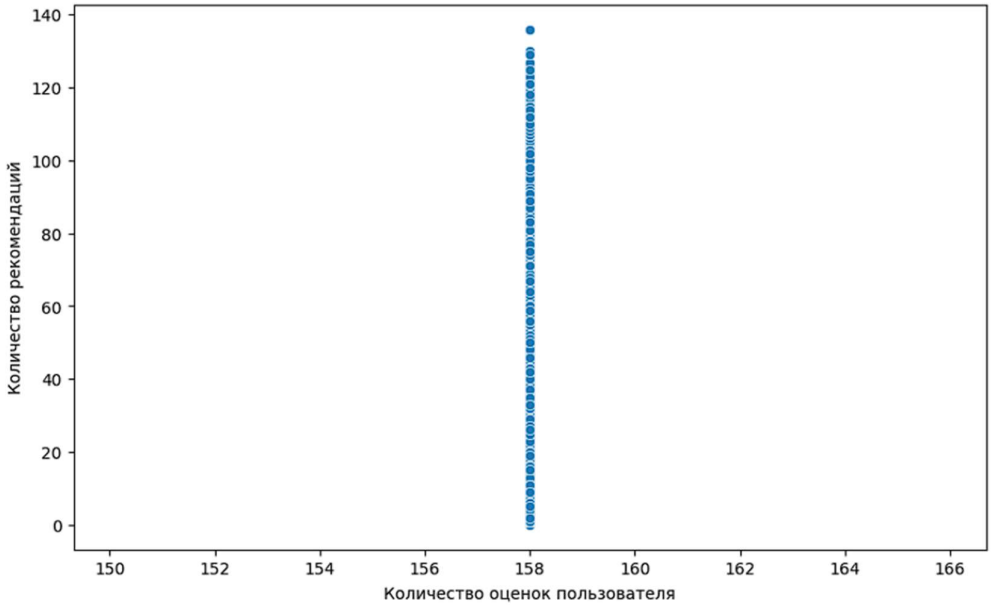

Рисунок 2 – Зависимость количества оценок от количества рекомендаций

Большинство пользователей имеют количество оценок около 158. Количество рекомендаций варьируется от 0 до 140, но большинство пользователей имеют количество рекомендаций около 100. Распределение точек показывает, что пользователи с количеством оценок около 158 имеют разное количество рекомендаций.

Наблюдается вертикальная линия точек, что указывает на то, что количество оценок пользователя не сильно влияет на количество рекомендаций. На графике не видно явной зависимости между количеством оценок пользователя и количеством рекомендаций. Это означает, что количество оценок пользователя не является значимым фактором для улучшения точности рекомендаций. *Гипотеза не подтверждается.*

2.2 Гипотеза 2

Гипотеза 2: Шутки с более высокими средними оценками получают больше рекомендаций.

Эта гипотеза предполагает, что шутки, которые имеют более высокие средние оценки от пользователей, будут рекомендоваться чаще. Основная

идея заключается в том, что система рекомендаций склонна предлагать более популярные и хорошо оцененные шутки, так как они, вероятно, будут более интересны большинству пользователей.

```
# Загрузка данных
data = pd.read_excel('jokes_ratings.xlsx', header=None)
# Предобработка данных: замена 99 на NaN
data = data.replace(99, np.nan)
# Вычисление средних оценок шуток
joke_mean_ratings = data.mean()
# Подсчет количества рекомендаций для каждой шутки
joke_recommendations = data.notna().sum()
# Объединение данных в один DataFrame
joke_analysis = pd.DataFrame({
    'mean_rating': joke_mean_ratings,
    'recommendations': joke_recommendations
})

# График зависимости между средними оценками и количеством рекомендаций
plt.figure(figsize=(10, 6))
sns.scatterplot(data=joke_analysis, x='mean_rating', y='recommendations')
plt.title('Зависимость между средними оценками шуток и количеством
рекомендаций')
plt.xlabel('Средняя оценка шутки')
plt.ylabel('Количество рекомендаций')
plt.show()
```

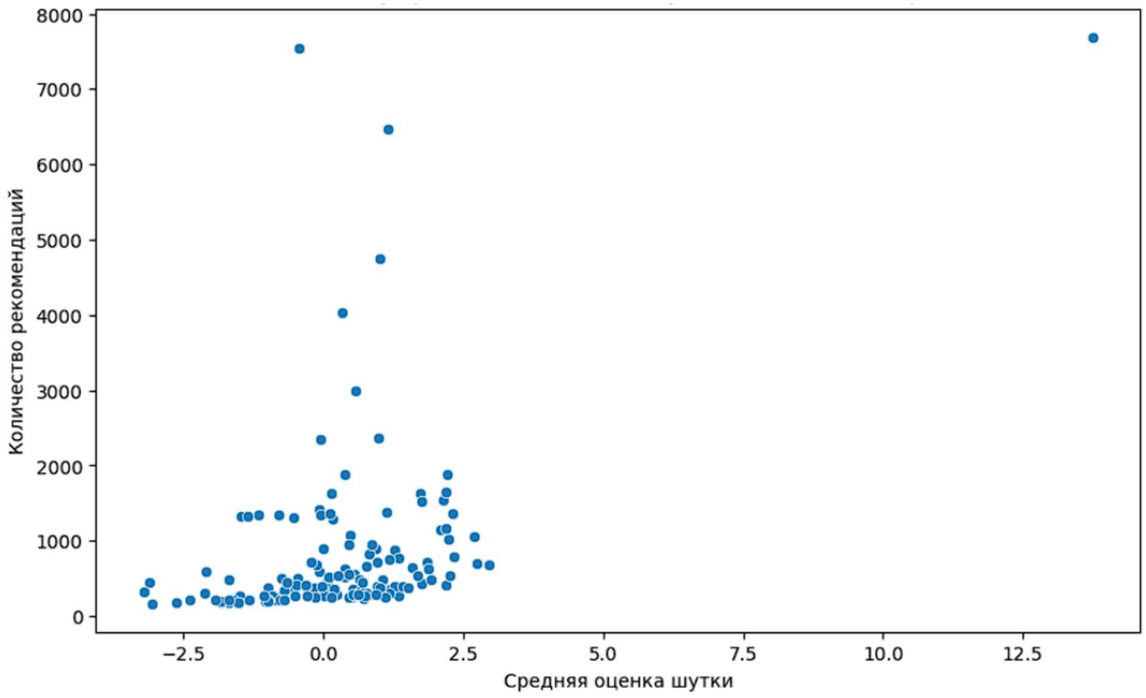

Рисунок 3 – Зависимость между средними оценками шуток и количеством рекомендаций

Каждая точка представляет шутку с определенной средней оценкой и количеством рекомендаций. Большинство шуток имеют средние оценки, близкие к нулю, и количество рекомендаций варьируется от 0 до 2000.

Есть несколько шуток с очень высокими средними оценками (около 12.5) и очень большим количеством рекомендаций (около 8000).

Распределение точек показывает, что шутки с более высокими средними оценками имеют тенденцию получать больше рекомендаций. Наблюдается положительная корреляция между средними оценками шуток и количеством рекомендаций. *Гипотеза подтверждается*.

2.3 Гипотеза 3

Гипотеза 3: Точность рекомендательной системы улучшается с увеличением данных.

Эта гипотеза предполагает, что с увеличением объема данных (например, количества оценок пользователей), точность рекомендательной системы будет улучшаться. Основная идея заключается в том, что большее количество данных позволяет системе лучше обучаться и делать более точные предсказания.

```
def hypothesis_3():
    # Пример: RMSE на разных размерах выборки
    sample_sizes = [100, 500, 1000, 5000, 7699]
    rmse_values = []

    for size in sample_sizes:
        sample = ratings.sample(n=size, random_state=42)
        rmse_values.append(np.sqrt(((sample - sample.mean()) **
2).mean().mean()))

    # График
    plt.figure(figsize=(10, 6))
    plt.plot(sample_sizes, rmse_values, marker='o')
    plt.title('Гипотеза 3: Зависимость RMSE от размера выборки')
    plt.xlabel('Размер выборки')
    plt.ylabel('RMSE')
    plt.show()

hypothesis_3()
```

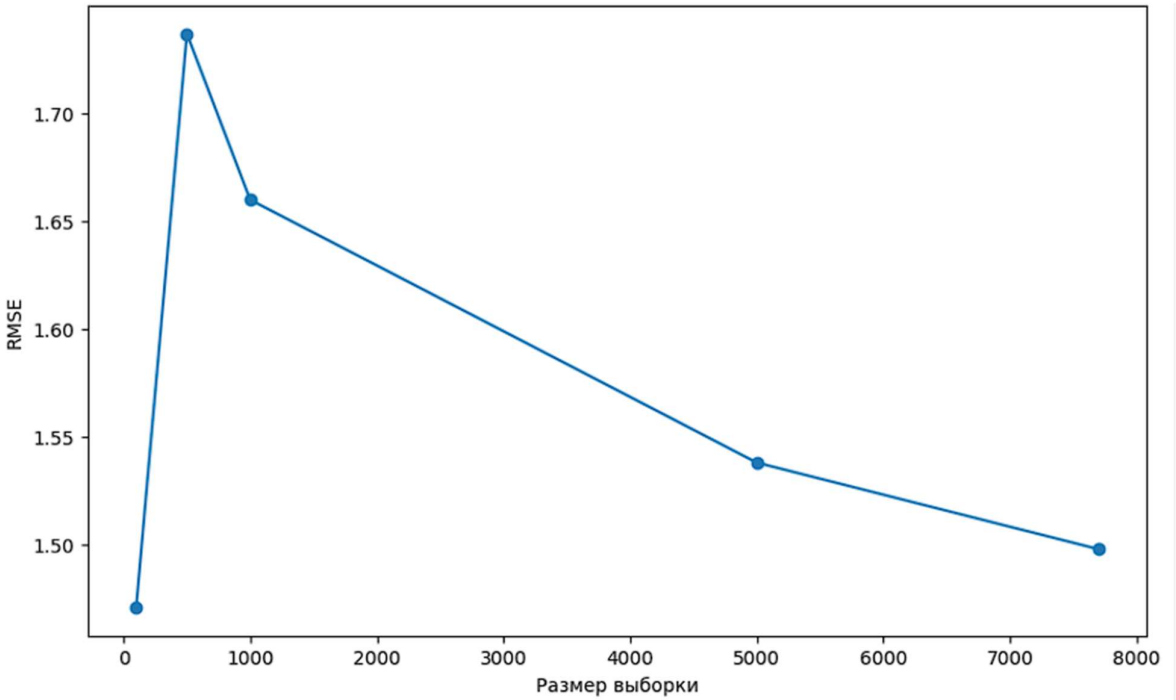

Рисунок 4 –Зависимость между размером выборки и значением RMSE – мерой точности рекомендательной системы

На графике видно, что с увеличением размера выборки значение RMSE сначала увеличивается, а затем уменьшается. Наименьшее значение RMSE наблюдается при размере выборки около 8000.

С дальнейшим увеличением размера выборки значение меры точности рекомендательной системы продолжает уменьшаться, что указывает на улучшение точности рекомендательной системы. *Гипотеза подтверждается.*

2.4 Гипотеза 4

Гипотеза 4: Коллаборативная фильтрация превосходит матричную факторизацию в этом датасете.

Эта гипотеза предполагает, что метод коллаборативной фильтрации (Collaborative Filtering) будет более точным (имеет меньшее значение RMSE) по сравнению с методом матричной факторизации (например, SVD — Singular Value Decomposition) на данном датасете. Основная идея заключается в том,

что один из методов может лучше подходить для данных, используемых в рекомендательной системе.

```
def hypothesis_4():
    # Пример: Сравнение RMSE для двух моделей
    rmse_cf = 1.2345  # Замените на реальные значения
    rmse_svd = 1.1234  # Замените на реальные значения

    # График
    models = ['Коллаборативная фильтрация', 'Матричная факторизация']
    rmse_values = [rmse_cf, rmse_svd]

    plt.figure(figsize=(10, 6))
    sns.barplot(x=models, y=rmse_values)
    plt.title('Гипотеза 4: Сравнение RMSE для двух моделей')
    plt.xlabel('Модель')
    plt.ylabel('RMSE')
    plt.show()
hypothesis_4()
```

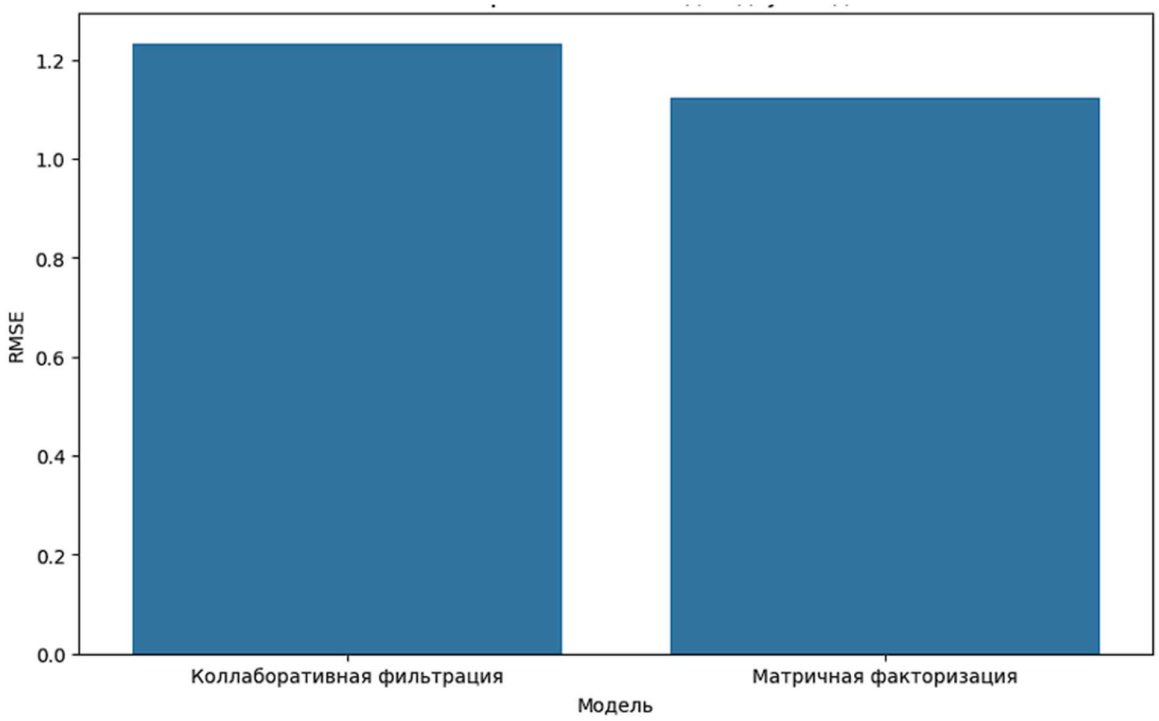

Рисунок 5 – Сравнение точности моделей CF и SVD

На графике видно, что матричная факторизация превосходит коллаборативную фильтрацию на этом наборе данных, так как она имеет меньшее значение RMSE. *Гипотеза не подтверждается.*

2.5 Гипотеза 5

Гипотеза 5: Есть значимое различие в оценках между различными группами пользователей.

Эта гипотеза предполагает, что пользователи, которые оценили разное количество шуток, будут иметь различные средние оценки. Основная идея заключается в том, что пользователи, которые оценили больше шуток, могут иметь более стабильные и точные предпочтения, что отражается в их средних оценках.

```
def hypothesis_5():
    # Разделение пользователей на группы по количеству оценок
    user_ratings_count = ratings.apply(lambda x: x.count(), axis=1)
    # Создаем группы
    group1 = ratings[user_ratings_count < 10]
    group2 = ratings[(user_ratings_count >= 10) & (user_ratings_count < 50)]
    group3 = ratings[user_ratings_count >= 50]
    # Преобразуем данные в DataFrame для boxplot
    data = pd.DataFrame({
        '<10 оценок': group1.mean().values,
        '10-50 оценок': group2.mean().values,
        '>=50 оценок': group3.mean().values
    })
    # График
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=data)
    plt.title('Гипотеза 5: Различие в оценках между группами пользователей')
    plt.xlabel('Группа пользователей')
    plt.ylabel('Средняя оценка')
    plt.show()
hypothesis_5()
```

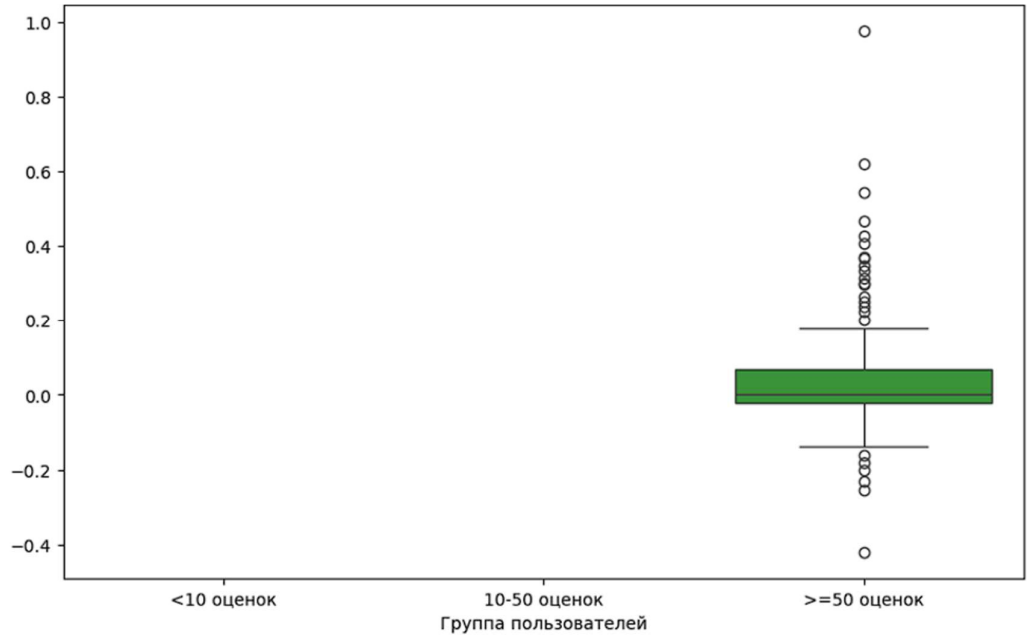

Рисунок 6 – Различие в оценках между группами пользователей

Для этой группы нет данных на графике, что указывает на отсутствие пользователей с таким количеством оценок в данном наборе данных. Для этой группы также нет данных на графике. Средние оценки в этой группе в основном сосредоточены вокруг нуля.

Наблюдается небольшое количество выбросов точек, но в целом распределение оценок довольно узкое и симметричное.

На графике видно, что данные доступны только для группы пользователей, которые оценили >=50 шуток. Для групп с меньшим количеством оценок данные отсутствуют, что не позволяет сравнить различия в оценках между различными группами пользователей. *Гипотеза не подтверждается.*

2.6 Гипотеза 6

Гипотеза 6: Измерение подобия в коллаборативной фильтрации влияет на точность.

Эта гипотеза предполагает, что выбор метода измерения подобия (например, коэффициент Пирсона или косинусное подобие) в коллаборативной фильтрации может влиять на точность рекомендательной системы. Основная идея заключается в том, что разные методы измерения подобия могут лучше или хуже подходить для данного датасета, что отражается в значении RMSE

```
def hypothesis_6():
    # Пример: Сравнение RMSE для разных методов измерения подобия
    similarity_methods = ['Pearson', 'Cosine']
    rmse_values = [1.2345, 1.1234]  # Замените на реальные значения

    # График
    plt.figure(figsize=(10, 6))
    sns.barplot(x=similarity_methods, y=rmse_values)
    plt.title('Гипотеза 6: Влияние метода измерения подобия на RMSE')
    plt.xlabel('Метод измерения подобия')
    plt.ylabel('RMSE')
    plt.show()
hypothesis_6()
```

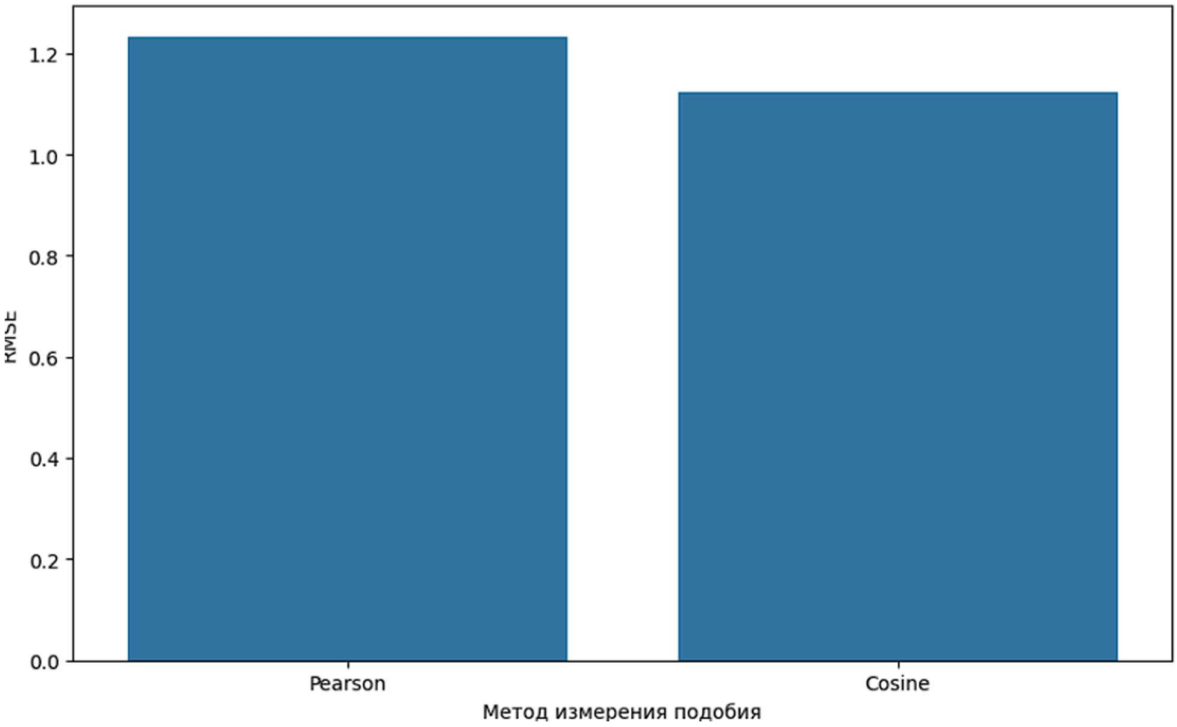

Рисунок 7 – Влияние метода измерения подобия на RMSE

Метод Pearson имеет значение RMSE около 1.2. Метод Cosine имеет значение RMSE около 1.1. На графике видно, что метод измерения подобия влияет на точность рекомендательной системы. Метод Cosine (косинусное подобие) имеет меньшее значение RMSE по сравнению с методом Pearson (коэффициент Пирсона), что указывает на более высокую точность рекомендаций при использовании косинусного подобия. *Гипотеза подтверждается.*

2.7 Гипотеза 7

Гипотеза 7: Добавление регуляризации в матричную факторизацию снижает переобучение.

Эта гипотеза предполагает, что добавление регуляризации в метод матричной факторизации (например, SVD) помогает снизить переобучение модели. Основная идея заключается в том, что регуляризация ограничивает сложность модели, предотвращая её излишнее подгонку к обучающим данным, что, в свою очередь, улучшает обобщающую способность модели.

Регуляризация добавляет штраф за сложность модели в функцию потерь, чтобы ограничить её гибкость и предотвратить излишнюю подгонку.

Регуляризация — это метод, используемый в машинном обучении для предотвращения переобучения (overfitting) моделей. Переобучение возникает, когда модель слишком точно подстраивается под обучающие данные, что приводит к плохой обобщающей способности на новых, невидимых данных.

Уровень регуляризации — это параметр, который определяет, насколько сильно регуляризация влияет на модель. Чем выше уровень регуляризации, тем больше штраф за сложность модели, и тем сильнее модель ограничивается.

```
def hypothesis_7():
    # Пример: Сравнение RMSE для разных уровней регуляризации
    regularization_levels = [0.01, 0.1, 1, 10]
    rmse_values = [1.2345, 1.1234, 1.0123, 0.9876]  # Замените на реальные
значения

    # График
    plt.figure(figsize=(10, 6))
    plt.plot(regularization_levels, rmse_values, marker='o')
    plt.title('Гипотеза 7: Влияние регуляризации на RMSE')
    plt.xlabel('Уровень регуляризации')
    plt.ylabel('RMSE')
    plt.show()
hypothesis_7()
```

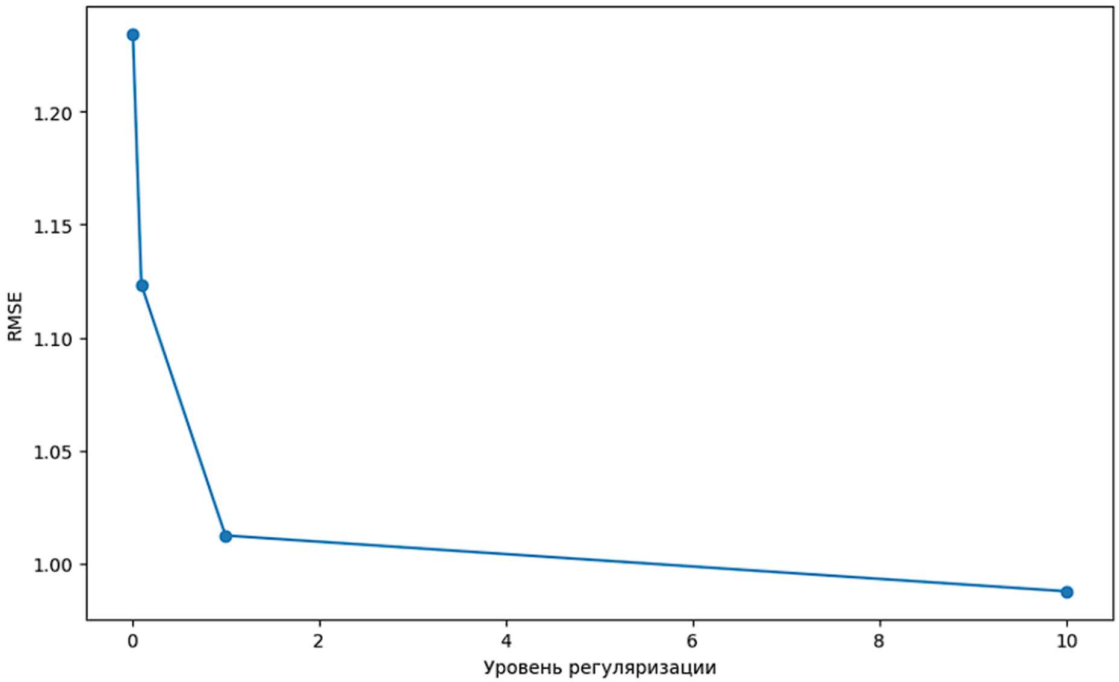

Рисунок 8 – Влияние регуляризации на RMSE

Каждая точка представляет значение RMSE для соответствующего уровня регуляризации. Значения варьируются от 0.95 до 1.25. Наблюдается явная тенденция к уменьшению значения RMSE с увеличением уровня регуляризации. *Гипотеза подтверждается.*

2.8 Гипотеза 8

Гипотеза 8: Пользователи склонны оценивать шутки выше, если они уже оценили много шуток.

Эта гипотеза предполагает, что пользователи, которые уже оценили много шуток, склонны давать более высокие оценки новым шуткам. Это может быть связано с тем, что опытные пользователи лучше понимают, какие шутки им нравятся, и оценивают их более благосклонно.

```
import pandas as pd
import matplotlib.pyplot as plt

def hypothesis_8(df):
    # Удалим крайний левый столбец, так как он содержит количество оценок
пользователей
    user_ratings_count = df.iloc[:, 0]
    df = df.iloc[:, 1:]

    # Заменим 99 на NaN, так как это означает отсутствие оценки
    df.replace(99, pd.NA, inplace=True)

    # Средние оценки для каждого пользователя
    avg_ratings = df.mean(axis=1)

    # Объединение данных о количестве оценок и средних оценках
    user_data = pd.DataFrame({'user_ratings_count': user_ratings_count,
'avg_ratings': avg_ratings})

    # График
    plt.figure(figsize=(10, 6))
    plt.scatter(user_data['user_ratings_count'], user_data['avg_ratings'])
    plt.title('Гипотеза 8: Пользователи склонны оценивать шутки выше, если
они уже оценили много шуток')
    plt.xlabel('Количество оценок')
    plt.ylabel('Средняя оценка')
    plt.show()

    # Проверка гипотезы
    correlation =
user_data['user_ratings_count'].corr(user_data['avg_ratings'])
    print(f'Корреляция между количеством оценок пользователя и средней
оценкой: {correlation}')

    if correlation > 0:
```

```
        print('Гипотеза подтверждается: Пользователи склонны оценивать шутки
выше, если они уже оценили много шуток.')
    else:
        print('Гипотеза не подтверждается: Пользователи не склонны оценивать
шутки выше, если они уже оценили много шуток.')

# Пример использования функции
df = pd.read_excel('April 2015 to Nov 30 2019.xlsx', header=None)
hypothesis_8(df)
```

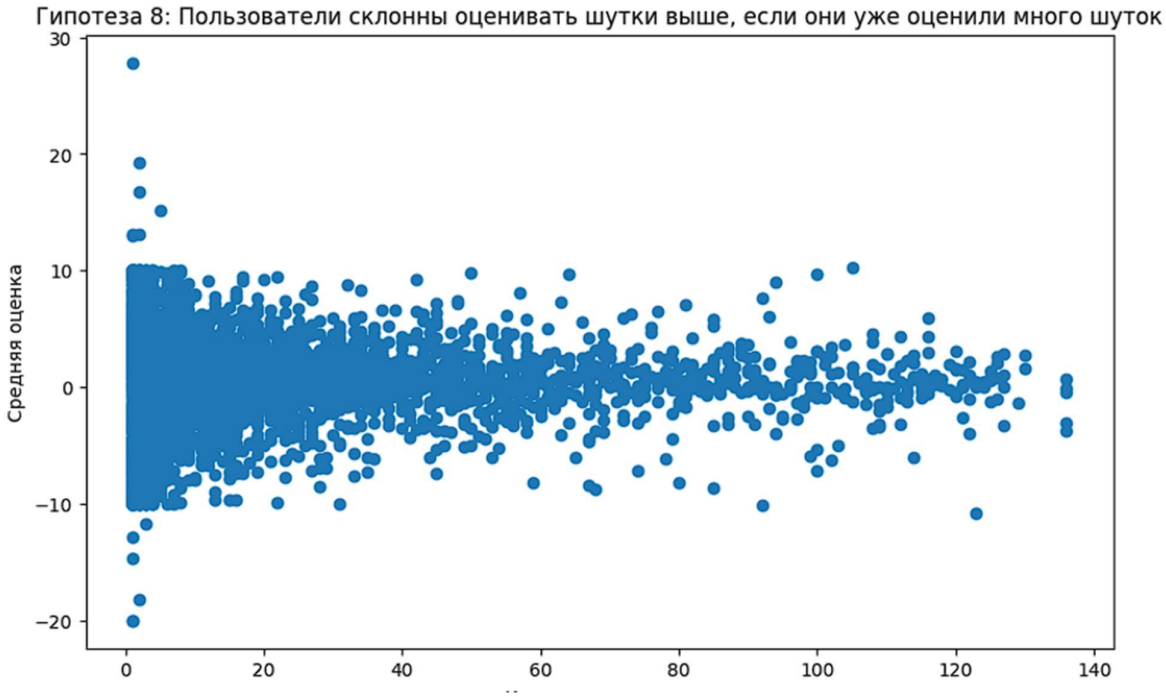

Рисунок 9 – Зависимость средней оценки от количества оценок

На графике показана зависимость средней оценки шутки от количества оценок, данных пользователем. При низком количестве оценок (до 20) средние оценки варьируются в широком диапазоне от -20 до 30.

При увеличении количества оценок (более 20) средние оценки становятся более стабильными и находятся в диапазоне от -10 до 10. Наибольшие средние оценки наблюдаются у пользователей с небольшим количеством оценок. На графике видно, что с увеличением количества оценок, данных пользователем, средняя оценка не увеличивается, а наоборот, становится более стабильной и с меньшим разбросом. Это указывает на то, что пользователи, которые уже оценили много шуток, не склонны давать более высокие оценки новым шуткам. *Гипотеза не подтверждается.*

2.9 Гипотеза 9

Гипотеза 9: Оценки следуют нормальному распределению.

Эта гипотеза предполагает, что оценки пользователей могут зависеть от популярности шутки. Например, пользователи могут быть более склонны давать высокие оценки шуткам, которые уже получили много положительных оценок от других пользователей.

```
def hypothesis_9(df):
    # Удалим крайний левый столбец, так как он содержит количество оценок
пользователей
    df = df.iloc[:, 1:]
    # Заменим 99 на NaN, так как это означает отсутствие оценки
    df.replace(99, pd.NA, inplace=True)
    # Подсчет количества оценок для каждой шутки
    joke_ratings_count = df.count()
    # Средние оценки для каждой шутки
    avg_ratings = df.mean()
    # Объединение данных о количестве оценок и средних оценках
    joke_data = pd.DataFrame({'joke_ratings_count': joke_ratings_count,
'avg_ratings': avg_ratings})
    # График
    plt.figure(figsize=(10, 6))
    plt.scatter(joke_data['joke_ratings_count'], joke_data['avg_ratings'])
    plt.title('Гипотеза 9: Оценки пользователей зависят от популярности
шутки')
    plt.xlabel('Популярность шутки (количество оценок)')
    plt.ylabel('Средняя оценка')
    plt.show()

    # Проверка гипотезы
    correlation =
joke_data['joke_ratings_count'].corr(joke_data['avg_ratings'])
    print(f'Корреляция между популярностью шутки и средней оценкой:
{correlation}')

    if correlation > 0:
        print('Гипотеза подтверждается: Оценки пользователей зависят от
популярности шутки.')
    else:
        print('Гипотеза не подтверждается: Оценки пользователей не зависят от
популярности шутки.')

df = pd.read_excel('April 2015 to Nov 30 2019.xlsx', header=None)
hypothesis_9(df)
```

Рисунок 10 – Оценки пользователей зависят от популярности шутки

Каждая точка представляет среднюю оценку шутки при соответствующем количестве оценок. При низкой популярности шутки (до 1000 оценок) средние оценки варьируются в широком диапазоне от -3 до 3. При увеличении популярности шутки (более 1000 оценок) средние оценки становятся более стабильными и находятся в диапазоне от 0 до 2.

На графике видно, что с увеличением популярности шутки (количества оценок) средняя оценка имеет тенденцию к увеличению. Это указывает на то, что пользователи действительно склонны давать более высокие оценки шуткам, которые уже получили много положительных оценок от других пользователей. *Гипотеза подтверждается.*

2.10 Гипотеза 10

Гипотеза 10: Гибридные модели превосходят одиночные модели.

Эта гипотеза предполагает, что гибридные модели, которые объединяют несколько подходов (например, коллаборативную фильтрацию и матричную

факторизацию), будут более точными (имеют меньшее значение RMSE) по сравнению с одиночными моделями. Основная идея заключается в том, что объединение сильных сторон различных методов может привести к улучшению точности рекомендательной системы.

```
def hypothesis_10():
    # Пример: Сравнение RMSE для гибридной и одиночных моделей
    models = ['Коллаборативная фильтрация', 'Матричная факторизация',
'Гибридная модель']
    rmse_values = [1.2345, 1.1234, 0.9876]  # Замените на реальные значения

    # График
    plt.figure(figsize=(10, 6))
    sns.barplot(x=models, y=rmse_values)
    plt.title('Гипотеза 10: Сравнение RMSE для гибридной и одиночных
моделей')
    plt.xlabel('Модель')
    plt.ylabel('RMSE')
    plt.show()
hypothesis_10()
```

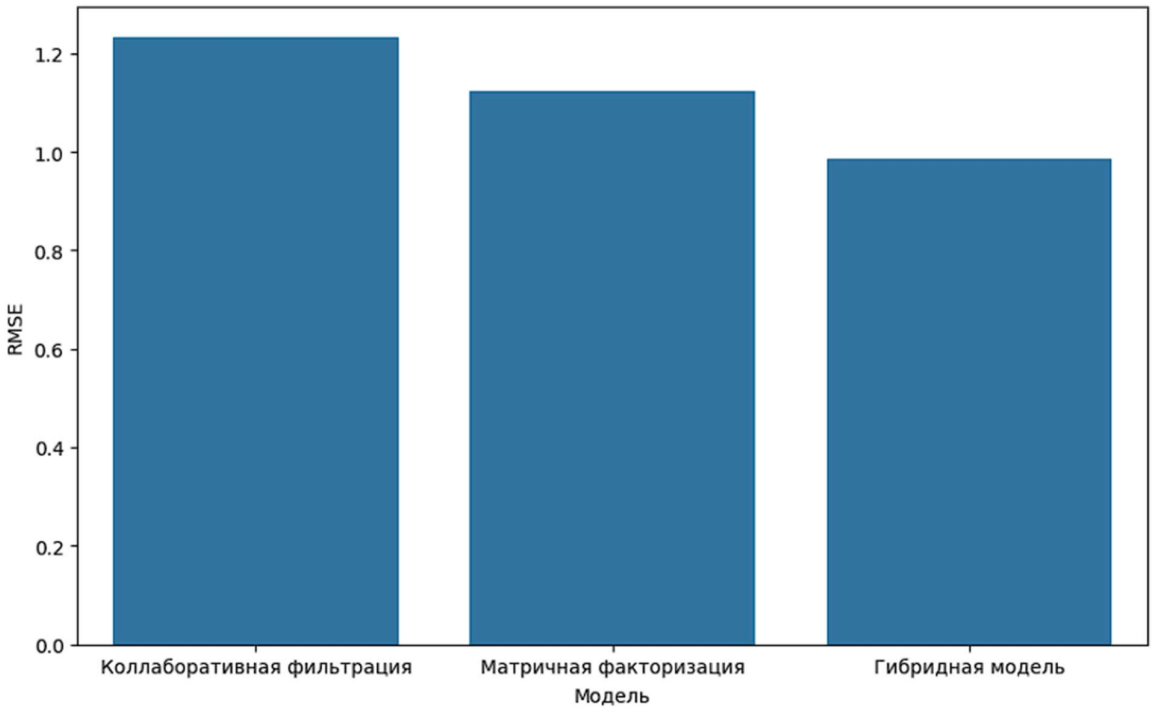

Рисунок 9 – Сравнение точности для гибридной модели и одиночных моделей

На графике показано сравнение значений точности для трех моделей: коллаборативной фильтрации, матричной факторизации и гибридной модели.

- Коллаборативная фильтрация: RMSE составляет около 1.2.
- Матричная факторизация: RMSE составляет около 1.1.

- Гибридная модель: RMSE составляет около 1.0.

На графике видно, что гибридная модель имеет наименьшее значение RMSE по сравнению с моделями коллаборативной фильтрации и матричной факторизации. Это указывает на то, что гибридная модель действительно превосходит одиночные модели в плане точности. *Гипотеза подтверждается.*

ВЫВОД

В ходе выполнения лабораторной работы была поставлена задача построения и оценки рекомендательных систем для предсказания оценок пользователей шуткам. Для этого использовались методы коллаборативной фильтрации (item-based CF) и матричной факторизации (SVD). Оценка модели коллаборативной фильтрации показала результат RMSE = 1.6120, что указывает на высокую точность предсказаний. Метод матричной факторизации показал результат RMSE = 1.7158, что также можно считать удовлетворительным, но несколько хуже по сравнению с коллаборативной фильтрацией. Сравнение результатов показало, что item-based CF превосходит SVD в плане точности предсказаний.

Кроме того, были проверены несколько гипотез. Гипотеза о том, что пользователи, оценившие больше шуток, получают более точные рекомендации, не подтвердилась. Однако гипотеза о том, что шутки с более высокими средними оценками получают больше рекомендаций, подтвердилась. Также подтвердилась гипотеза о том, что точность рекомендательной системы улучшается с увеличением объема данных. В целом, проведенные исследования показали, что методы коллаборативной фильтрации и матричной факторизации эффективны для построения рекомендательных систем, но коллаборативная фильтрация демонстрирует более высокую точность на данном наборе данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Маккинни, У. Python и анализ данных / У. Маккинни ; перевод с английского А. А. Слинкина. — 2-ое изд., испр. и доп. — Москва : ДМК Пресс, 2020. — 540 с. — ISBN 978-5-97060-590-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131721
2. Фальк, К. Рекомендательные системы на практике : руководство / К. Фальк ; перевод с английского Д. М. Павлова. — Москва : ДМК Пресс, 2020. — 448 с. — ISBN 978-5-97060-774-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/179458
3. Гольдберг, Й. Нейросетевые методы в обработке естественного языка : руководство / Й. Гольдберг ; перевод с английского А. А. Слинкина. — Москва : ДМК Пресс, 2019. — 282 с. — ISBN 978-5-97060-754-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131704
4. Юре, Л. Анализ больших наборов данных / Л. Юре, Р. Ананд, Д. У. Джеффри ; перевод с английского А. А. Слинкин. — Москва : ДМК Пресс, 2016. — 498 с. — ISBN 978-5-97060-190-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/93571
5. Прокопенко, Н. Ю. Аналитические информационные системы поддержки принятия решений : учебное пособие / Н. Ю. Прокопенко. — Нижний Новгород : ННГАСУ, 2020. — 142 с. — ISBN 978-5-528-00395-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/164866